Priority Queues and Heaps

CSE 373 Winter 2020

Instructor: Hannah C. Tang

Teaching Assistants:

Aaron Johnston	Ethan Knutson	Nathan Lipiarski
Amanda Park	Farrell Fileas	Sam Long
Anish Velagapudi	Howard Xiao	Yifan Bai
Brian Chan	Jade Watkins	Yuma Tou
Elena Spasova	Lea Quan	

Poll Everywhere

pollev.com/uwcse373

About how long did Homework 3 take?

A. 0-2 Hours
B. 2-4 Hours
C. 4-6 Hours
D. 6-10 Hours
E. 10-14 Hours
F. 14+ Hours
G. I haven't finished yet / I don't want to say

Announcements

- Homework 4: Heap is released and due *Wednesday*
  - Hint: you will need an additional data structure to improve the runtime for changePriority(). This data structure may or may not be a (classic) Red-Black tree.
- Workshop this Friday @ 11:30am, CSE 203
  - Topics include 2-3 Trees and LLRBs
- Please attend 373 DITs, not other classes'!

Questions from Reading Quiz

- When do we use Priority Queues?
- How is a Queue and Priority Queue different?
- How do we handle duplicate values?

Lecture Outline

- **Priority Queues and Review: Binary Trees**
- Binary Heaps
- Binary Heap Representation

ADTs So Far (1 of 3)

Set ADT. A collection of values.

- A set has a size defined as the number of elements in the set.
- You can add and remove values.
- Each value is accessible via a “get” or “contains” operation.

Map ADT. A collection of keys, each associated with a value.

- A map has a size defined as the number of elements in the map.
- You can add and remove (key, value) pairs.
- Each value is accessible by its key via a “get” or “contains” operation.

ADTs So Far (2 of 3)

List ADT. A collection storing an ordered sequence of elements.

- Each element is accessible by a zero-based index.
- A list has a size defined as the number of elements in the list.
- Elements can be added to the front, back, *or any index in the list*.
- Optionally, elements can be removed from the front, back, *or any index in the list*.

ADTs So Far (3 of 3)

Deque ADT. A collection storing an ordered sequence of elements.

- Each element is accessible by a zero-based index.
- A deque has a size defined as the number of elements in the deque.
- Elements can be added to the front or back.
- Optionally, elements can be removed from the front or back.

Stack ADT. A collection storing an ordered sequence of elements.

- A stack has a size defined as the number of elements in the stack.
- Elements can only be added and removed from the top (“LIFO”)

Queue ADT. A collection storing an ordered sequence of elements.

- A queue has a size defined as the number of elements in the queue.
- Elements can only be added to one end and removed from the other (“FIFO”)

We found more-performant data structures to implement the Queue ADT when we took advantage of its more-limited-than-list functionality

ADTs To Come

Priority Queue ADT. A collection of values.

- A PQ has a size defined as the number of elements in the set.
- You can add values.
- You cannot access or remove arbitrary values, only the max value.

Today's Topic!

Disjoint Se on!

- After erm

Graph A Coming S

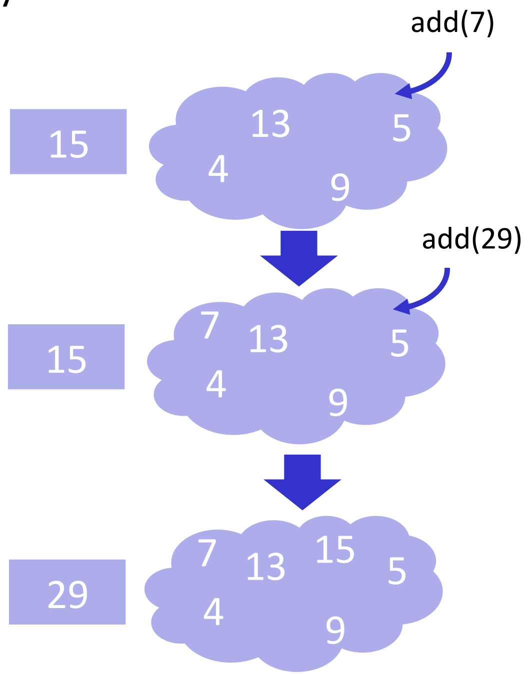
- After th

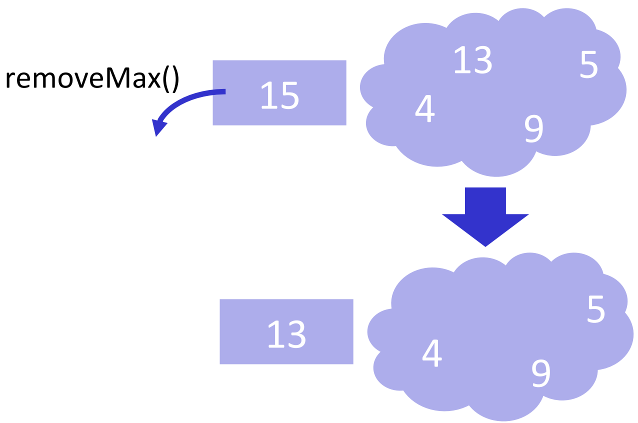
Soon, but not yet!

Can we find a more-performant data structure to implement the Priority Queue ADT when we take advantage of its more-limited-than-queue functionality?

Priority Queues

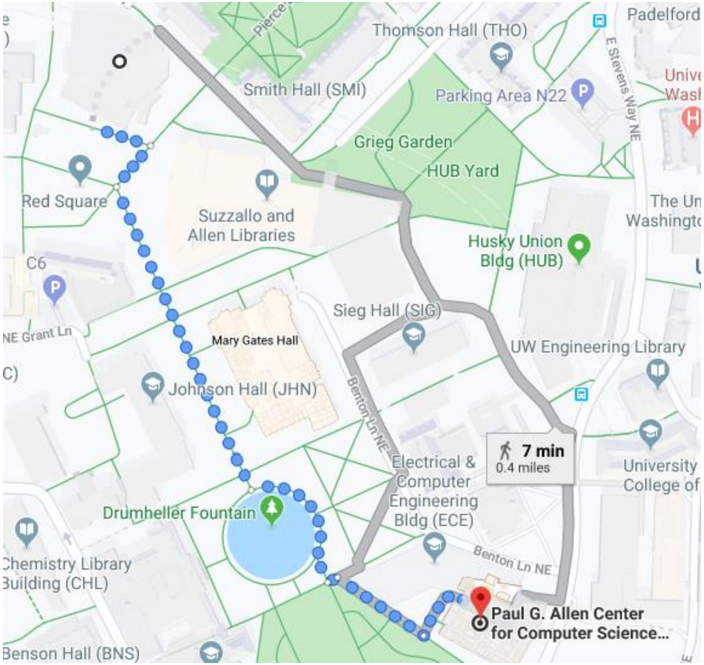
- In lecture, we will study **max priority queues** but **min priority queues** are also common
  - Same as max-PQs, but invert the priority
- In a PQ, the only item that matters is the max (or min)





Priority Queue: Applications

- Used heavily in **greedy algorithms**, where each phase of the algorithm picks the locally optimum solution
- Example: route finding
  - Represent a map as a series of *segments*
  - At each intersection, ask which segment gets you closest to the destination (ie, has max priority or min distance)

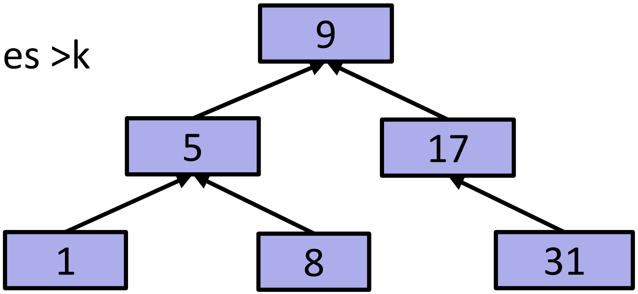


Lecture Outline

- Priority Queues and Review: Binary Trees
- **Binary Heaps**
- Binary Heap Representation

Review: Binary *Search* Trees

- A **Binary Search Tree** is a binary tree with the following invariant: for every node with value k in the BST:
  - The left subtree only contains values <k
  - The right subtree only contains values >k




```
class BSTNode<Value> {
  Value v;
  BSTNode left;
  BSTNode right;
}
```

Reminder: the BST ordering applies recursively to the entire subtree

Priority Queue: Possible Data Structures

- We have two viable implementations of this ADT (so far):

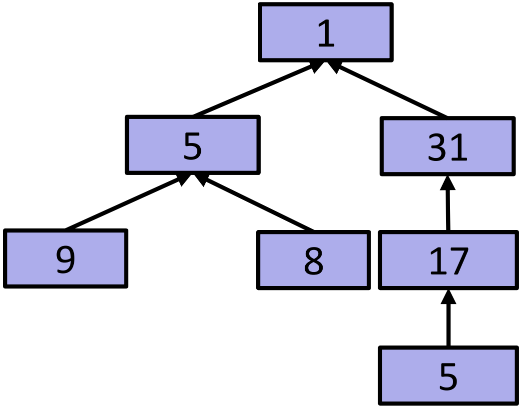
	Sorted LinkedList PQ (worst case)	Balanced Search Tree PQ (worst case)
add	O(N)	O(log N)
max	O(1)	O(1)*
removeMax	O(1)	O(log N)

** If we keep a pointer to the largest element in the BST*

Review: Binary Tree Data Structure

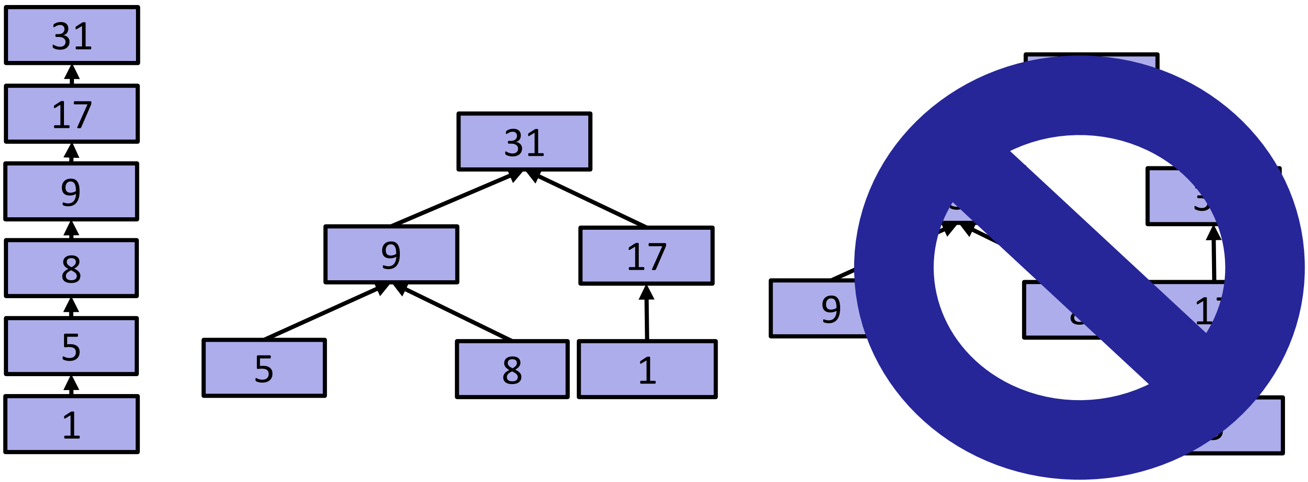
- A **Binary Tree** (not a binary *search* tree) is a tree where each node has 0 <= children <= 2

```
class BinaryNode<Value> {
  Value v;
  BinaryNode left;
  BinaryNode right;
}
```



Heaps

- A **Max Heap**: a binary tree where each node's value is greater than any of its descendents. It implements the Max Priority Queue ADT
  - This is a *recursive* property!
- A **Min Heap** is the same, but each node is *less than* its descendents



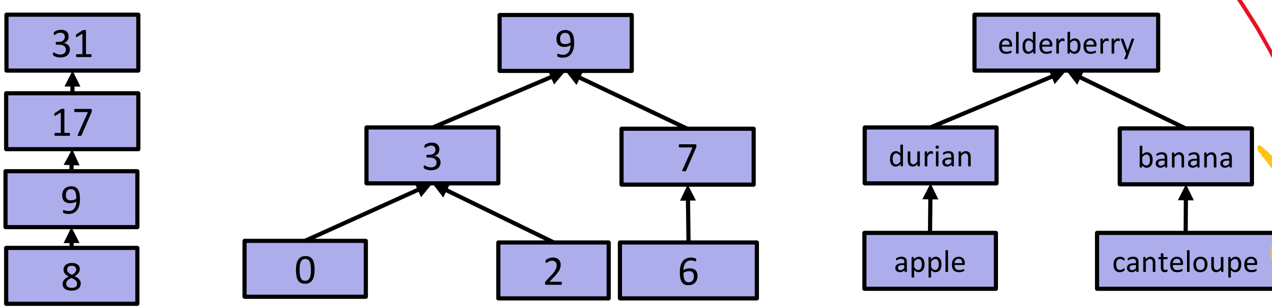
Poll Everywhere

pollev.com/uwcse373

- Which of these are valid max heaps?

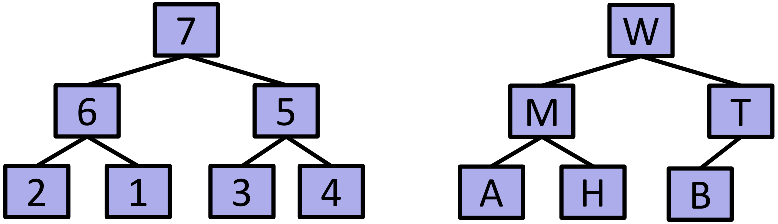
1. Valid / Invalid / Valid
2. Valid / Invalid / Invalid
3. Valid / Valid / Invalid
4. Valid / Valid / Valid

violation of (max) heap invariant



Binary Heaps

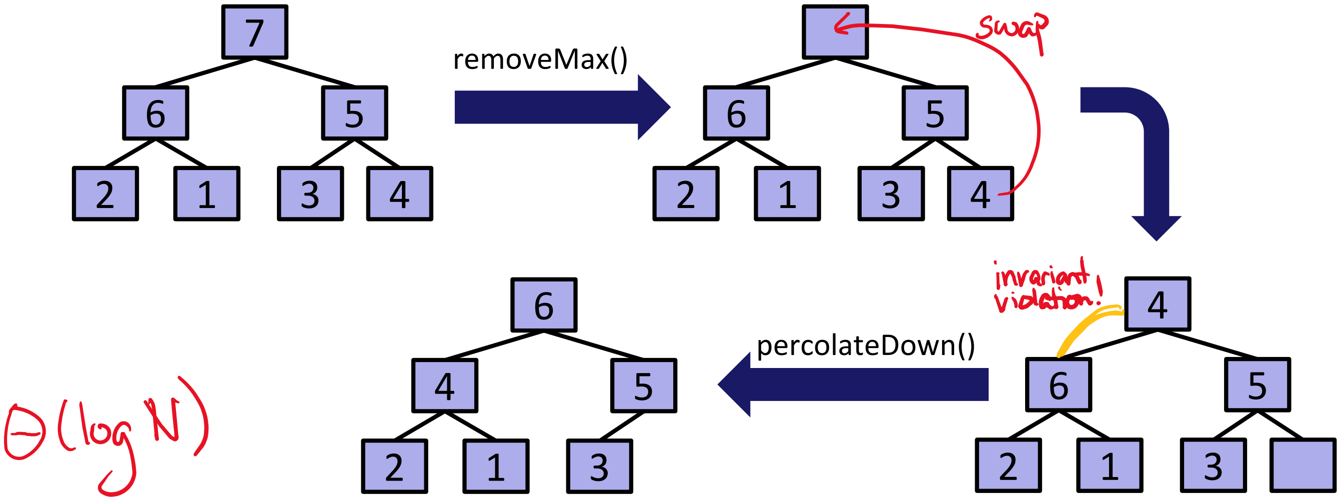
- A **Binary Heap** is a heap that is completely filled, with the possible exception of the bottom level which is filled left-to-right
  - Its height is Θ(log N)



but is removeMax and add also ∈ Θ(log N)?

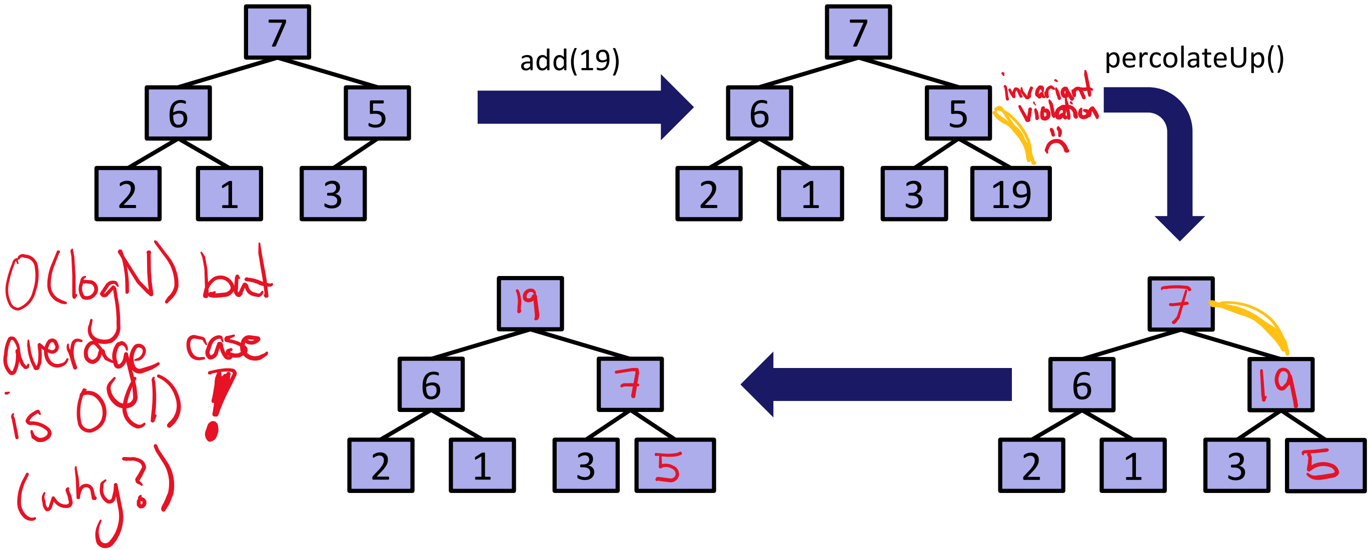
Binary Heaps: removeMax

- Remove the root's value (but keep the root node)
- Swap in the to-be-deleted leaf's value
- Recursively percolateDown() against each level's larger child



Binary Heaps: add

- Add the new value at the next valid location in the complete tree
- Recursively percolateUp() … ?

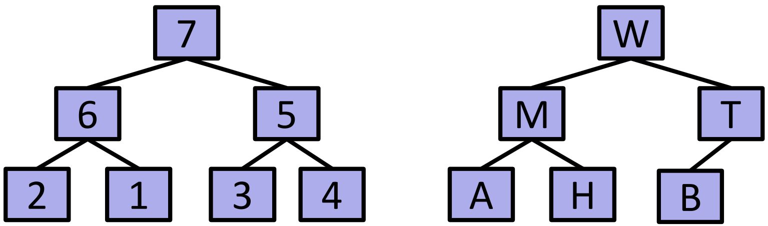


Lecture Outline

- Priority Queues and Review: Binary Trees
- Binary Heaps
- **Binary Heap Representation**

Binary Heaps as Arrays

- A **Binary Heap** is a heap that is completely filled, with the possible exception of the bottom level which is filled left-to-right

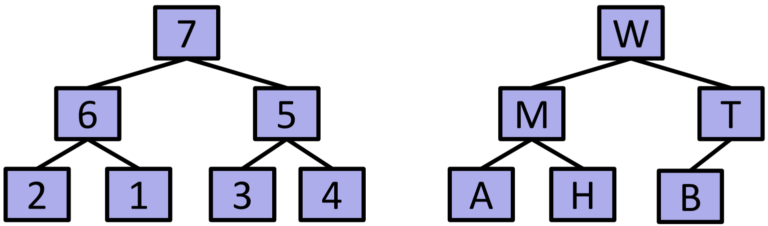


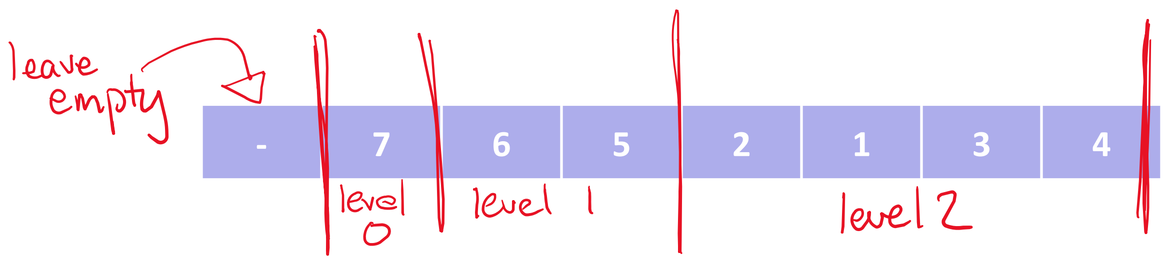
- … which makes it easily representable as an array
  - (note: we leave the 0th index empty to make the arithmetic easier)

-	7	6	5	2	1	3	4
-	W	M	T	A	H	B	

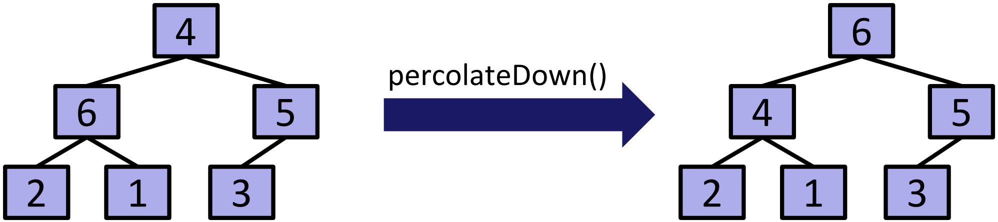
Binary Heaps as Arrays

- A **Binary-Heap-as-Array**'s node with index i has:
  - Its children at 2*i and 2*i + 1
  - Its parent at i/2





Binary Heaps as Arrays: percolateDown



-	4	6	5	2	1	3	

```
void percolateDown(int idx) {
  tmp = a[idx];
  for ( ; idx * 2 <= a.length; ) {
    idx = idx * 2;
    if (a[idx] < a[idx + 1]) idx++;
    if (a[idx] > tmp) {
      a[idx/2] = a[idx];
    } else {
      break;
    }
  }
}
```

Find our children in the array

Get the index of our larger child

We've rewritten our recursive algorithm iteratively!

swap if we're still violating the heap invariant

Other Priority Queue Operations

- The two "primary" PQ operations are:
  - removeMax()
  - add()
- However, because PQs are used in so many algorithms there are three common-but-nonstandard operations:
  - merge(): merge two PQs into a single PQ
  - buildHeap(): reorder the elements of an array so that its contents can be interpreted as a valid binary heap
  - changePriority(): change the priority of an item already in the heap

we'll revisit soon!

you will implement in HW4!

Other Priority Queue data structures

- **D-Heaps**
  - Binary heap, but with a >2 branching factor “d”
- **Leftist Heap**
  - Unbalanced heap that skews “leftward”, optimized for merge()
- **Skew Heap**
  - Leftist Heap variant, also optimized for merge()
- **Binomial Queue**
  - A “forest” of heaps

tl;dr

- **Priority Queue ADT** is designed to find the max (or min) quickly
  - We can implement it with many data structures

- **The Binary Heap** is a data structure which is simple to reason about and implement *and* has constant- to Θ(log N) bounds

	Sorted LL (worst case)	Balanced BST (worst case)	Binary Heap (worst case)
add	O(N)	O(log N)	O(log N)**
max	O(1)	O(1)*	O(1)
removeMax	O(1)	O(log N)	O(log N)

** If we keep a pointer to the largest element in the BST*
*** Average case is constant*

BONUS! ADT / Data Structure Taxonomy

ADT

Maps and Sets

Data Structures that Implement

- Search Trees ("left is less-than, right is greater-than")
  - Binary Search Trees (branching factor == 2)
    - **Plain BST** (unbalanced)
      - Balanced BSTs: **LLRB** (other examples: **"Classic" Red-Black**, **AVL**, **Splay**, etc)
  - B-Trees (have a branching factor >2; balanced)
    - **2-3 Trees**
    - **2-3-4 Trees**
- Hash Tables (will cover later!)